\*References: Appendix-C

**Criterion C: Product Development**

Techniques used to create the program (discussed later on):

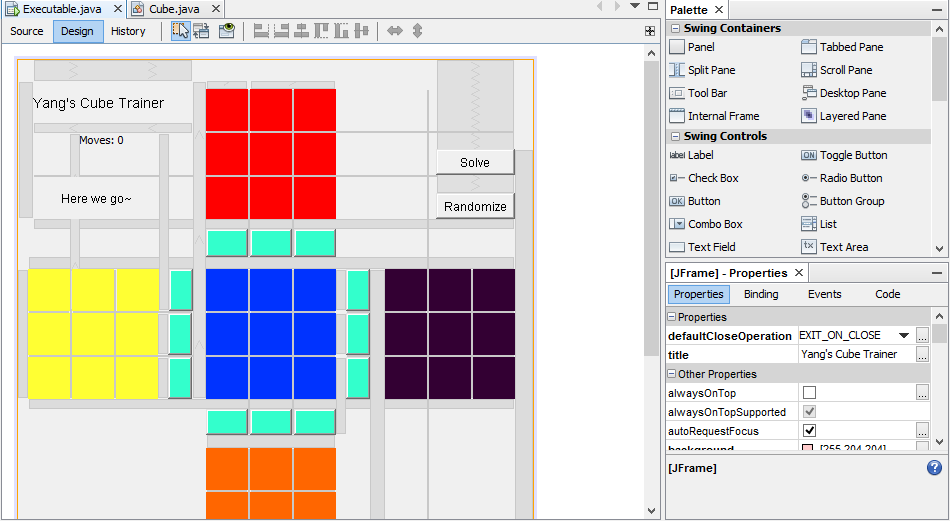
1. Graphics to create a visual user interface as text-based outputs are really inefficient as well as hard to understand by the user (especially in terms of color)
   1. Panels which change color in order to reflect the faces of the cube
   2. Clickable buttons to rotate the cube faces among other interactions
   3. Textbox to inform the user if they’ve completed it and how many moves they’ve made
2. An ArrayList as the main data structure to store the previous and (if re-winded) future stages of the cube so the user can access any move, anytime, anywhere. There are also other data structures
3. An additional cube class is to be created just for simplicity and better organization of code as well as making it easier to analyze for mistakes
4. A real life Rubik’s Cube was extremely important in creating this program to help as it helps as a physical means to discover the rotatory patterns of the faces myself

Techniques to minimize errors:

1. Create the graphics interface with the cube solved, this way I will always start with the simplest arrangement so I can test new functions and whatnot
2. Frequent testing. I always code in sections where the program will be able to compile freely and I can troubleshoot on the way instead of just during the final stage

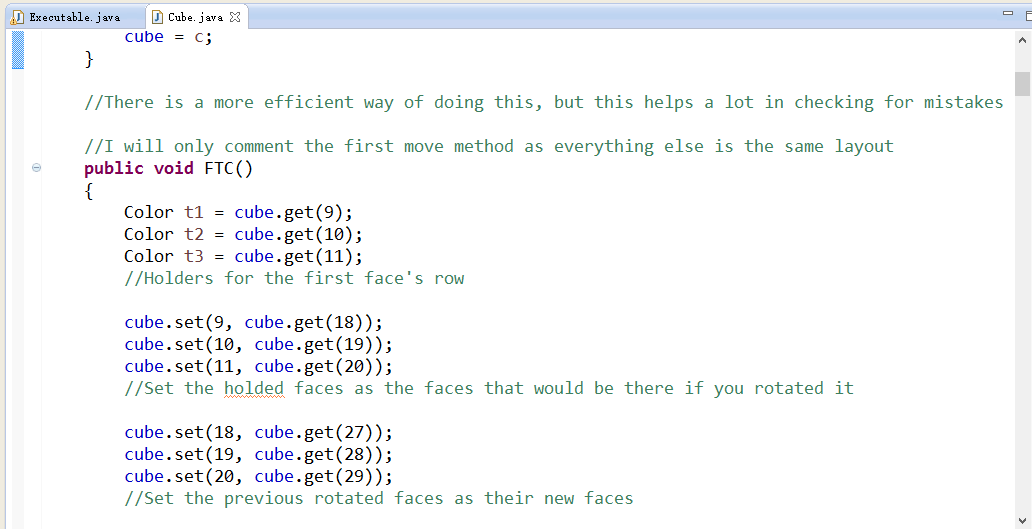
Explanations of Techniques:

1. Graphics interfaces
   1. Netbeans was used to assist in creating the visual portions of the program
      1. Provides a visual plan so the developer can make small tweaks very quickly, no constant reruns to test changes
      2. Drag and drop means less work and saves time from learning new fields for me, as I have not touched upon graphics code for all that much
   2. Netbeans is only used for the graphics interface, the coding is done with Eclipse, a sample of the discussed elements is screen-shotted below
      1. Easily change color
      2. Guide lines to assist with lining up tiles and buttons
      3. Drag and drop interfaces very simple to arrange



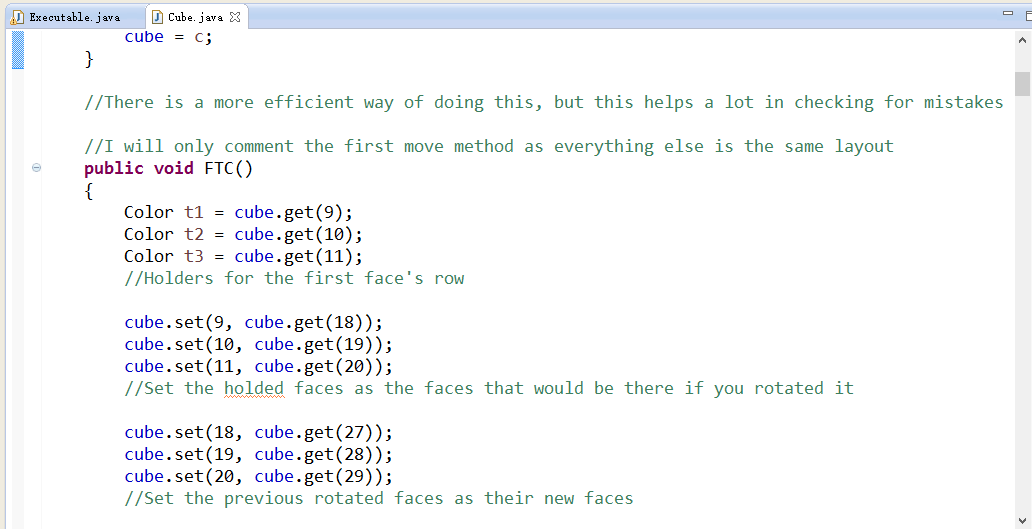
* 1. Able to create a skeleton structure code, with all the names and initialization already set so I can just go in focus on what the things actually do
  2. Overall utilized for the code because it saves a lot of time compared to actually programming the graphics myself. I can then focus on perfecting my algorithms and how to organize and run the code smoother. It’s as functional as if I had done it myself, but it’s important because it indirectly makes the other segments better

1. Databases (ArrayList)
   1. The array list contains all the past and present positions of the cube, colors for the cube itself, and other lists. It is functionally amazing to have as it can do so much, helps greatly with programming
      1. Easy access to particular positions when pushing forward or rewinding, just increments/decrements
      2. It can store objects. In this case, the cube
      3. It can store other Array Lists. The colors for each face of the cube must be stored individually, helps greatly with organization and cutting down on code
      4. Functionally infinite positions to be utilized
   2. The list gets cleared when the user randomizes the positions for another round, very convenient as it only takes a single line of code to clear it
   3. Other information storage includes
      1. Storing in the class. Each face of the cube is stored inside the class itself as it can simply be retrieved by 1 increment instead of 54 increments of faces from the master list. Easier to read/organize code as well as runs smoother
      2. Certain values are created to keep count of moves and the like. These are just incremented and decremented as needed in the main class
   4. Overall utilized extensively throughout the exploration because it is easy to use, hold a lot of functionality for what I wish to achieve, and it’s something I have a lot of experience working with already. It’s extremely high flexibility and very usable functions for my purposes makes it an ideal choice for the main data storage/structure in my code
2. Cube Class
   1. The separate class contains all of the methods for the cube
      1. All my cube code is gathered in one place, if an error occurs and I know it’s a functionality problem, it narrows down the search area, helping with trouble shooting as well as reasons discussed later
   2. Multiple cubes are created in the main class, this cuts down on a lot of copy and pasted code, it also helps with organization since I have Array List of cube objects instead of a list with the colors list and being forced to equate more temporary place holders for accessing past/future cube positions
      1. Easier to read and edit
      2. Localized trouble-shooting means if an error occurs, I only have to fix a single chunk instead of locating the same chunk copy pasted throughout the whole program, well laid out as shown below



* 1. Overall selected for use in programming because of my previous experience with separate classes. It cuts down on a lot of space and it localizes trouble shooting as well as code reading both in its own class and in the main executable file. Because it helps with organization, cuts down on required lines of code, and provides easier access it is a very important selection in my coding process

1. Real Life Rubik’s Cube
   1. Had drawn out the cube on paper and numbered all the faces a different number
      1. See development plan (Figure 1.) on Page 5
      2. Helps me with figuring out the rotational patterns
   2. Used the real cube to model out which number goes where in order to program the replacement code. Using the same picture as above, but for a different purpose, I had to do the face colors swapping literally by holding onto colors and rotating them manually (as there was no better way), it helped greatly with logic



* 1. If you see the code above, it is definitely very confusing and without the real cube and the planning sheet, it would not have been possible. As it was basically necessary for the successful creation of the program, this is one of the major techniques I have chosen to include in this discussion

1049 Words